

Nro. ord.	Apellido y nombre	L.U.	#hojas
			7

ORGANIZACIÓN DEL COMPUTADOR I - Parcial

Segundo Cuatrimestre 2019 - Tercer Año

Ej.1	Ej.2	Ej.3	Ej.4	Nota
B	B	B	B	A

Corrector:

Aclaraciones

- Anote apellido, nombre, LU y numere todas las hojas entregadas
- Cada ejercicio se califica con Bien, Regular o Mal. La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se califican globalmente.
- El parcial **NO** es a libro abierto, pero puede tener los apuntes provistos por la cátedra y una hoja A4 con apuntes propios
- **Importante:** Justifique sus respuestas. Las soluciones a ejercicios de la práctica que se utilicen deben ser incluidas en el examen.
- El parcial se aprueba con un Bien y a lo sumo un Regular en los ejercicios 1, 2. En los ejercicios 3 y 4 al menos uno Bien.

Ejercicio 1 La LEM es una máquina con arquitectura Von Neumann que opera con palabras e instrucciones de tamaño fijo y utiliza aritmética en complemento a 2. Tiene 64 registros de propósito general y una memoria direccionable a *byte* con direcciones de 10 *bits* y con palabras de 16 *bits*. El tamaño del dato de la unidad de memoria es de 16 *bits*.

El set de instrucciones es el siguiente:

Instrucción	Formato	opCode	Efecto
MOV regX, regY	RR	0000	$regX \leftarrow regY$
ADD regX, regY	RR	0001	$regX \leftarrow regX + regY$
AND regX, regY	RR	0010	$regX \leftarrow regX \& regY$ (bit a bit)
INC regX	R	0011	$regX \leftarrow regX + 1$
JZ inm	RJ	0100	si $Z = 1$, $PC \leftarrow PC + ext(inm)$
JN inm	RJ	0101	si $N = 1$, $PC \leftarrow PC + ext(inm)$
LD reg, inm	RM	0110	$reg \leftarrow [inm]$
ST reg, inm	RM	0111	$[inm] \leftarrow reg$
JMP inm	M	1000	$PC \leftarrow inm$

En todos los casos, la referencia a PC corresponde al valor del mismo al realizar la ejecución de la instrucción. *ext()* se refiere a extender el signo hasta el tamaño adecuado. Las únicas instrucciones que afectan los flags son ADD y NEG. LD y ST sólo utilizan R0 a R3.

Las instrucciones tienen 4 tipos de formatos posibles:

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	4 bits				6 bits						6 bits					
RR	opCode				reg						reg					
	4 bits				6 bits						6 bits					
R	opCode				0						reg					
	4 bits				4 bits				8 bits							
RJ	opCode				0				inm							
	4 bits				2 bit		10 bits									
RM	opCode				reg		inm									
	4 bits				2 bit		10 bits									
M	opCode				0		inm									

a. Definir (justificando su respuesta):

- El tamaño del PC
- El tamaño máximo de la memoria
- Cuántas instrucciones sin operandos podría agregar manteniendo las instrucciones actuales (Tenga cuidado que las instrucciones actuales sigan siendo posible de decodificar).
- Considerando que una instrucción inválida es una tira de *bits* que no codifica una instrucción que la máquina puede ejecutar, determinar de las siguientes codificaciones cuáles lo son: 0x29, 0xF1F0, 0x7777, 0x6000.

- b. Se quiere ensamblar y cargar desde la posición de memoria 0x000 el siguiente código en una LEM:

```
inicio: LD R0, 0x0018
        LD R1, 0x0002
        AND R0, R1
        INC R0
        ADD R0, R0
seguir: INC R0
        CMP R1, R0
        JZ fin
        JMP seguir
dato:   DW 0x8016
fin:    JMP dato
        DW 0x0000
```

- I. Definir a qué posición de memoria corresponde cada etiqueta.
 - II. Indicar el contenido de la memoria luego de ensamblar y cargar el código anterior.
- c. Para cada parte de la planilla de seguimiento de la máquina ORGA1, justificar si es necesaria o no dicha celda. Indicar si es necesario agregar nuevas celdas o no para poder realizar el seguimiento.
- d. Modificar la planilla de seguimiento de la ORGA1 y realizar el seguimiento de la LEM. Tomar en cuenta que los registros de propósito general empiezan en cero y la memoria se carga inicialmente totalmente en cero.
- e. Mostrar el contenido de la memoria luego de la ejecución del programa.

Ejercicio 2

- a. Realizar el circuito del componente INC, que toma un valor de 16 bits y devuelve el mismo valor incrementado en una unidad.
- b. Realizar el diagrama del *datapath* de una microarquitectura para la LEM que soporte la ejecución de las instrucciones descritas. Recuerde indicar el tamaño de cada registro, de los buses internos y externos, las señales de cada componente y justificar la utilización de cada componente escogido y cada decisión tomada.

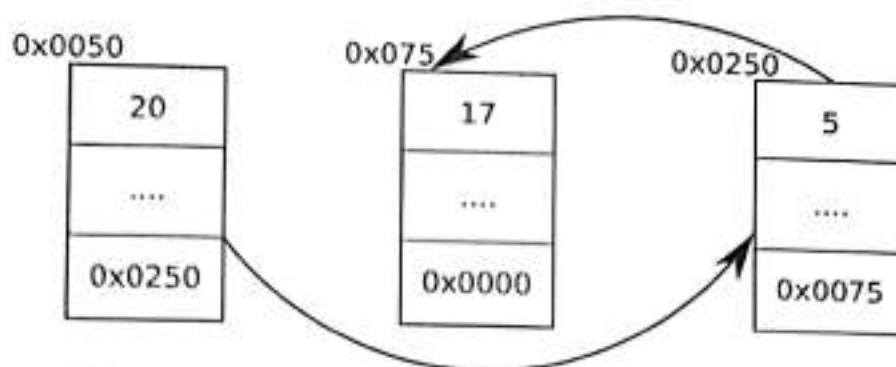
Para realizar el *datapath* puede utilizar los siguientes componentes:

- una única ALU que realiza las operaciones suma y resta **con flags**.
- extensores de signo y componentes para completar con ceros. (Deberá indicar claramente el tamaño de los registros de entrada y salida)
- un único controlador de memoria.
- el componente del apartado anterior (Si no lo realizó puede suponerlo hecho)

Al incluirlos detallar cuidadosamente el tamaño de los registros y los nombres de las señales. Cualquier otro componente a utilizar deberá ser implementado e incluido en el examen.

- c. Escribir las microinstrucciones que debe ejecutar la máquina para realizar el *fetch* de una instrucción (no incluir etapas posteriores del ciclo de instrucción).
- d. Escribir el microprograma que realiza la parte de ejecución del ciclo de instrucción de las siguientes instrucciones:
 - I. INC R0
 - II. JC 0x8016
 - III. JMP 0x0000

Ejercicio 3 Se cuenta con una máquina ORGA1. Se cuenta con la siguiente estructura (se indica cada celda en qué posición de memoria se encuentra):



La misma es una lista enlazada en la que cada celda contiene un número de 16 bits que indica el tamaño de la carga útil del nodo en bytes, luego la carga útil (cuya longitud en bytes es igual a la indicada en el primer dato) y luego otro número de 16 bits referenciando al siguiente elemento en la lista. El último elemento de la misma será el que tiene su referencia al próximo nodo un valor de 0x0000.

Realizar una función que tome en R0 la posición de memoria del primer elemento de una estructura. La misma deberá recorrerla sumando en cada paso la longitud de la carga útil. Devolver el resultado en R1.

En el ejemplo dado, el número final deberá ser: $20 + 17 + 5 = 42$

Ejercicio 4 Un grupo de seguidores del Mal Encarnado Pero Cachivachero (MEPC) tiene intención de desenterrar a su líder, el demonio del hielo y la ensalada conocido como Carlitos Nomebanioperoacompanio, quien se encuentra sepultado bajo varias toneladas de roca de diverso tipo de dureza. La conocida empresaria y filántropa de medio tiempo Vicenta Yahoraqué ha sido contratada para diseñar una excavadora que le permita a MEPC llegar al sitio donde Carlitos Nomebanioperoacompanio se encuentra prisionero. La máquina ha de contar con un taladro de dos velocidades, un extractor de aire, un sensor de gases peligrosos y otro de torque. El taladro será controlado por una máquina ORGAl.

El esquema de control es el siguiente: si se detecta la presencia de gases peligrosos en el ambiente se debe detener el taladro y encender el extractor, si no se detecta la presencia de gases peligrosos y el sensor de torque indica que el taladro se encuentra trabajando por sobre su capacidad (valores mayores a 100), se deberá disminuir la velocidad (valor bajo o 1), en caso contrario, si el sensor de torque no indica sobrecarga, el taladro podrá operar en la velocidad nominal (valor 2).

Los dispositivos de E/S son detallados a continuación:

TALADRO: controla la velocidad de operación del taladro, un valor 0x0000 lo apaga, 0x0001 lo hace trabajar en baja velocidad y 0x0002 lo lleva a los valores nominales.

TORQUE: informa la fuerza que está siendo aplicada sobre el taladro, su rango de valores se encuentra entre 0 y 255, los valores mayores a 100 son considerados como sobrecarga.

EXTRACTOR: controla el extractor de aire, un valor 0x0000 lo apaga y 0xFFFF lo enciende.

Al cambiar el estado en relación a la presencia de gases peligrosos se INTR se pondrá en 1 y se mantendrá su valor hasta recibir un pulso en la entrada de un bit llamada INTA. Se debe suponer que al comienzo de la operación no hay gases peligrosos en el ambiente y que la comunicación entre el sensor de gases y el procesador ORGAl es confiable (no se pierden señales).

- Mapear los registro de E/S e indicar para cada uno si es de lectura, escritura o lectura/escritura.
- Realizar el pseudo-código de la rutina principal y de la RAI que se debe cargar en la máquina ORGAl para lograr el funcionamiento detallado.
- Realizar el código, en lenguaje ensamblador, de la rutina principal y de la RAI.

1)

2) i. El tamaño del PC es de 10 bits, dado que almacena la posición en memoria de la instrucción que vamos a ejecutar y el enumerado dice que las direcciones de la memoria son de 10 bits (el programa se encuentra cargado en memoria por ser arquitectura Von Neumann).

ii. Como tenemos direcciones de 10 bits, hay 2^{10} direcciones de memoria posibles, donde en cada posición podemos guardar un byte (8 bits) por enumerado, por lo que el tamaño máximo de la memoria es de 2^{10} bytes. ✓

iii. De los 16 códigos de operación posibles (2^4 por la A de 4 bits para el op. code) hay 9 en uso, por lo que tengo 7 posibilidades para los primeros 4 bits y el resto está libre, así que en principio puedo agregar $7 \cdot 2^8$ instrucciones nuevas.

$7 \cdot 2^{12}$

Además, las instrucciones de tipo R, RJ y M tienen bits en 0 y puedo aprovechar esos bits para agregar más instrucciones.

Entonces nos queda:

la cuenta se hizo antes de agregar CMP

$$(7 \cdot 2^8) + (2^6 - 1) + (2 \cdot (2^4 - 1)) + (2^2 - 1)$$

$R = 2^6$ Rj M

En los 3 tipos de instrucciones calculo todas las combinaciones posibles menos la que tiene todos en cero. En Rj multiplico por 2 debido a que hay 2 Opcode de este tipo.

iv.

- $0x29$ tiene 8 bits y las instrucciones son de 16. No es válida. /
- $0xF1F0 = 1111\ 0001\ 1111\ 0000$. Los primeros cuatro bits no codifican un Opcode válido. /
- $0x7777 = \underbrace{0111}_{\text{opcode}} \underbrace{0111}_{\text{reg}} \underbrace{0111\ 0111}_{\text{imm}}$. Es una instrucción válida de tipo RM. /
- $0x6000 = \underbrace{0110}_{\text{opcode}} \underbrace{0000\ 0000\ 0000}_{\text{reg imm}}$. Es una instrucción válida de tipo RM. /

b)

- i.
- | | | | |
|----------|---------|---|--|
| inicio : | $0x000$ | → | pos inicial según enunciado |
| seguir : | $0x00A$ | → | hay 5 instrucciones entre inicio y seguir y cada una ocupa 2 pos. en memoria. $5 \cdot 2 = 10 = 0xA$ |
| dato : | $0x017$ | / | |
| fin : | $0x018$ | / | |

SIGUE EN LA PROX. HOJA

ii.	etiqueta	instrucción	codificación binaria	cod hexa	pos en memoria
	inicio	LD R0, 0x0018	0110 0000 0001 1000	0x6018	0x000
		LD R1, 0x0002	0110 0100 0000 0010	0x6102	0x002
		AND R0, R1	0010 0000 0000 0001	0x2001	0x004
		INC R0	0011 0000 0000 0000	0x3000	0x006
		ADD R0, R0	0001 0000 0000 0000	0x1000	0x008
	seguir	INC R0	0011 0000 0000 0000	0x3000	0x00A
		CMP R1, R0	1001 0000 0100 0000	0x9040	0x00C
		JZ Fin	0100 0000 0000 0110	0x4006	0x00E
		JMP seguir	1000 0000 0000 1010	0x800A	0x010
		DW 0x8016	1000 0000 0001 0110	0x8016	0x012
	dato	JMP dato	1000 0000 0001 0000	0x8014	0x014
	Fin	DW 0x0000	0000 0000 0000 0000	0x0000	0x016

JZ 'fin' reemplazo por pos etiqueta - PC post fetch de la instr. actual
 = JZ 0x016 - 0x010 = JZ 0x006

JMP 'seguir' y JMP 'dato' reemplazo los etiquetas directamente

- c)
- PC : necesario (hay que saber en donde está la instrucción).
 - SP : no es necesario (no hay stack).
 - R0..R7 : para este programa necesitamos solo R0 y R1. Ⓢ
 - flags : solamente necesitamos Z y N por los saltos condicionales.
 - memoria : necesario para cargar el programa.

Ⓢ El LEM tiene 64 registros, incluso programamos más podrían interesar agregarlos.

Además, como el direccionamiento es a bytes y el programa se carga desde una posición por, hecho todas las celdas impares para no tener que estar reparando las instrucciones en 2 (ya que la palabra es de 16 y una celda solo guarda 8). \otimes

d) En la planilla de seguimiento. Luego de la última instrucción decodificada saltamos a seguir (0x00A) y corramos de vuelta esas instrucciones hasta que R0 llegue al valor de R1 que es 0x6102 y ahí termina el programa.

e) En la planilla de seguimiento.

\otimes Como las palabras son de longitud fija /
siempre necesitamos la 2da o 3era palabra.

0x0001 → 0x0002
0x6102

Planilla de Seguimiento

Valores iniciales:

PC	SP
0x000	

R0	R1	R2	R3	R4	R5	R6	R7
0x0000	0x0000						

Z	C	V	N
0			0

Memoria:

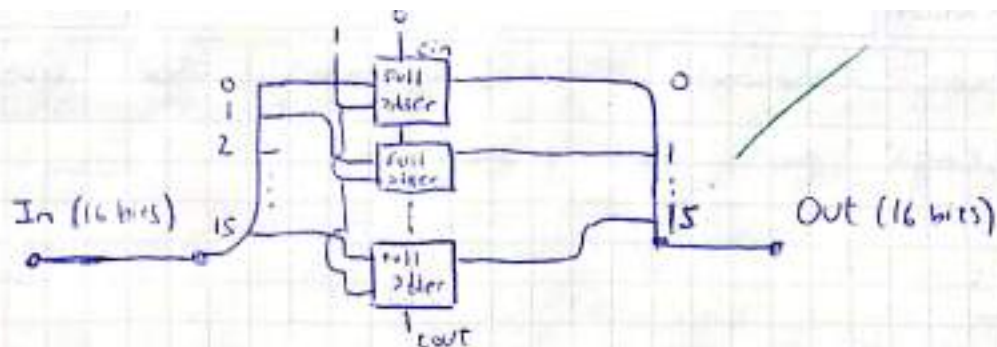
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0x000	0x6018		0x6102		0x2001		0x3000		0x1000		0x7000		0x6000		0x4000	
0x010	0x800A		0x8016		0x8014		0x0000		0x0000		0x0000		0x0000		0x0000	

	PC	SP	[SP+1]	IR	Instrucción - 1 ^{er} palabra (en bits)	PC	2 ^{da} palabra	PC	3 ^{ra} palabra	PC	Instrucción decodificada				
1	0x000			0x6018	0110 0000 0001 1000	0x002					LD R0, 0x018				
	Ejecución $R0 := [0x018] = 0x0000$										Flags ¹	Z	C	V	N
2	0x002			0x6102	0110 0100 0000 0010	0x004					LD R1, 0x002				
	Ejecución $R1 := [0x002] = 0x6102$										Flags ¹	Z	C	V	N
3	0x004			0x2001	0010 0000 0000 0001	0x006					AND R0, R1				
	Ejecución $R0 := R0 \text{ and } R1 = 0x6102 \text{ and } 0x0000 = 0x0000$										Flags ¹	Z	C	V	N
4	0x006			0x3000	0011 0000 0000 0000	0x008					INC R0				
	Ejecución $R0 := R0 + 0x0001 := 0x0001$										Flags ¹	Z	C	V	N
5	0x008			0x1000	0001 0000 0000 0000	0x00A					ADD R0, R0				
	Ejecución $R0 := R0 + R0 = 0x0001 + 0x0001 = 0x0002$										Flags ¹	Z	C	V	N
6	0x00A			0x3000	0011 0000 0000 0000	0x00C					INC R0				
	Ejecución $R0 := R0 + 0x0001 = 0x0003$										Flags ¹	Z	C	V	N
7	0x00C			0x9040	1001 0000 0100 0000	0x00E					CMP R1, R0				
	Ejecución Activación flags según R1 - R0										Flags ¹	Z	C	V	N
8	0x00E			0x4006	0100 0000 0000 0110	0x010					JZ fin				
	Ejecución Nada porque el flag Z sigue activo										Flags ¹	Z	C	V	N

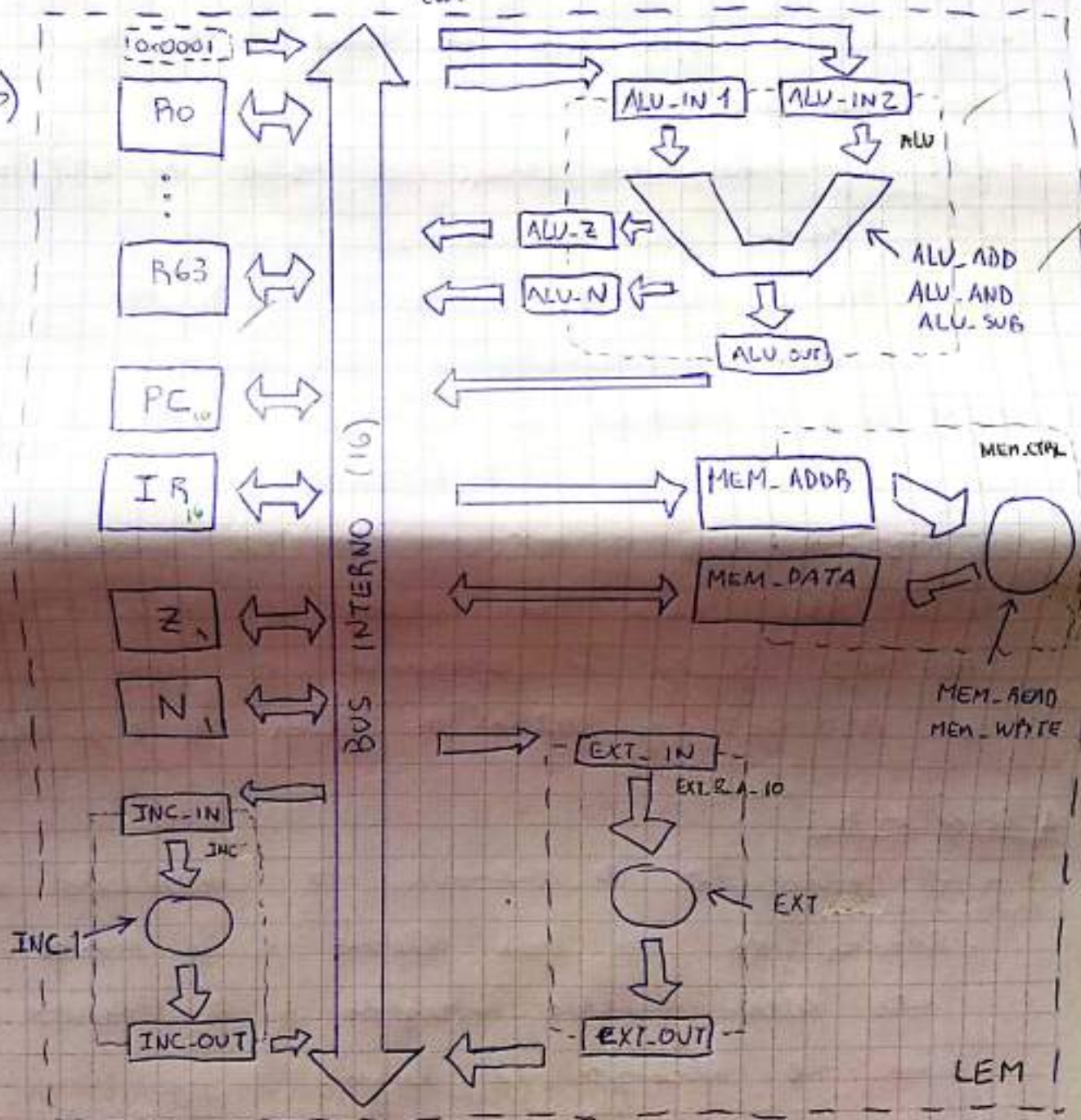
¹ Sólo deben completarse luego de la ejecución de una instrucción que los active.

2)

a)



b)



- Los registros, el IR, los ALU IN y OUT, MEM-DATA son de 16 bits.
- El PC, los INC-IN y INC-OUT, EXT-OUT y MEM-ADDR

- son de 10 bits y se encuentran conectados a las 10 líneas menos significativas del bus.
- Los flags son de 1 bit.

Señales:

- INC-1 le suma uno al valor en INC-IN y lo pone en INC-OUT.
- EXT extendiendo con ceros el valor de EXT-IN y lo coloca en EXT-OUT.
- MEM-READ lee la memoria en la posición MEM-ADDR y coloca el resultado en MEM-DATA.
- MEM-WRITE escribe en memoria en la pos. MEM-ADDR con lo que hay en MEM-DATA.
- ALU-ADD, ALU-SUB y ALU-AND son las operaciones típicas de la ALU, toman los valores de ALU-IN1 y ALU-IN2, ejecuta la operación y pone el resultado en ALU-OUT y actualiza flags (ALU-Z y ALU-N).

Aclaraciones:

- El controlador de memoria no está conectado a través de un bus externo a la memoria, esto debería haberlo agregado en el diagrama pero no me alcanzó el lugar.
- Los registros Z y N están para las operaciones de la ALU que actualizan flags, ya que hay algunas (ALU-AND por ejemplo) que no quieren que actualicen.

c) FETCH
 MEM-ADDR := PC
 MEM-READ
 IR := MEM-DATA
 INC-IN := PC
 INC-1
 PC := INC-OUT
 INC-IN := PC
 INC-1
 PC := INC-OUT

d) INC RO: ALU-IN1 := RO ESTÁ BIEN PERO
 ALU-IN2 := cre1 TENIAS UN COMPO
 ALU-ADD NGUTE INC.
 RO := ALU-OUT

JZ 0x8016: EXT-IN := IR[7:0]
 EXT
 ALU-IN1 := EXT-OUT
 ALU-IN2 := PC
 ALU-ADD
 PC := ALU-OUT

Me doy cuenta
 que hay un
 problema en el
 diseño, ahora
 atraí!

JMP 0x0000: PC := IR[9:0] ✓

Aclaración importante!

→ necesidad por copiar
lo - error.

Me di cuenta que tendría que agregar un elemento de 10 a 16 bits para poder sumas con la ALU y después poder guardar los 10 bits menos significativos del resultado en el PC.

Además me di cuenta de que el componente del item '3' era de 16 bits pero yo lo puse de 10 para el fetch y agregué una constante para el INC.

OK!

3)

Pseudo código:

nodoActual \leftarrow primero()
suma $\leftarrow 0$

While (nodoActual.siguiente() $\neq 0$)

suma \leftarrow suma + nodoActual.carga
nodoActual \leftarrow nodoActual.siguiente()

Implementación:

MOV R1, 0x0000 ; inicializo en 0

sumar: ADD R1, [R0]
ADD R0, [R0]
CMP R0, 0x0000
JE FIN

MOV R0, [R0]

JMP sumar

FIN: RET

BIEN SENCILLO!

4)

- a)
- TALADRO \mapsto 0xFFFO LECTURA?
 - TORQUE \mapsto 0xFF1 ESCRITURA?
 - EXTRACTOR \mapsto 0xFF2 AMBOS?

b)

```
principal () {  
    if (TORQUE > 100) EXTRACTOR.set(0)  
        TALADRO.setVelocidad(1)  
    else while (true) {  
        TALADRO.setVelocidad(2)  
    }  
}
```

rai () {

~~TALADRO.setVelocidad(0)~~
~~EXTRACTOR.set(1)~~

~~}~~

if (EXTRACTOR.is-set(0))

TALADRO.setVelocidad(0)

EXTRACTOR.set(1)

else if (EXTRACTOR.is-set(1))

~~TALADRO.setVelocidad(1)~~

EXTRACTOR.set(0)

c) $100 = 0110\ 0100 = 0x0064$

rutina ppa1:

MOV [0xFFF2], 0x0000

rutina: CMP [0xFFFF], 0x0064

JLE vel Normal

MOV [0xFFFF0], 0x0001

JMP rutina

vel Normal: MOV [0xFFFF0], 0x0002

JMP rutina

rai:

CMP [0xFFF2], 0x0000

JNE apagarExtractor

MOV [0xFFFF0], 0x0000

MOV [0xFFF2], 0xFFFF

JMP fin

apagarExtractor: MOV [0xFFF2], 0x0000

fin : IRET